1.CAESAR CIPHER:

```
#include <iostream>
#include <string>
using namespace std;
void encryptDecrypt(bool isEncrypt) {
   string text;
   int key;
   cout << "Enter a message: ";
   cin >> text;
   cout << "Enter key: ";
   cin >> key;
   for (char &ch : text) {
      char base = (ch >= 'a' && ch <= 'z') ? 'a' : (ch >= 'A' && ch <= 'Z') ? 'A' : 0;
      if (base) {
         ch = isEncrypt ? (ch + key - base) % 26 + base : (ch - key - base + 26) % 26 + base;
      }
   }
   cout << (isEncrypt ? "Encrypted" : "Decrypted") << " message: " << text << endl;
}
int main() {
   int choice;
   cout << "Choose an option:\n1. Encryption\n2. Decryption\n";
   cin >> choice;
   if (choice == 1)
      encryptDecrypt(true);
   else if (choice == 2)
      encryptDecrypt(false);
   else
      cout << "Invalid option!" << endl;
   return 0;
}
```

2.a.GCD:

```
#include <iostream>
using namespace std;
int main()
 {
  int n1, n2, gcd;
  cout << "Enter two numbers: ";
  cin >> n1 >> n2;
  if ( n2 > n1) {
    int temp = n2;
    n2 = n1;
    n1 = temp;
  }

  for (int i = 1; i <=  n2; ++i) {
    if (n1 % i == 0 && n2 % i ==0) {
      gcd = i;
    }
  }
cout << "GCD = " << gcd;
  return 0;
}
```

2.b.EXTENDED EUCLIDEAN:

```
#include <iostream>
using namespace std;

// Extended Euclidean Algorithm
int extendedEuclidean(int a, int b, int &x, int &y) {
   if (!b) {
      x = 1; y = 0;
      return a;
   }
   int x1, y1, gcd = extendedEuclidean(b, a % b, x1, y1);
   x = y1;
   y = x1 - (a / b) * y1;
   return gcd;
}

// Multiplicative Inverse
int multiplicativeInverse(int a, int m) {
   int x, y, gcd = extendedEuclidean(a, m, x, y);
   return (gcd == 1) ? (x % m + m) % m : -1;
}

int main() {
   int a, b, x, y;
   cout << "Enter two numbers (a, b): ";
   cin >> a >> b;

   int gcd = extendedEuclidean(a, b, x, y);
   cout << "GCD of " << a << " and " << b << " is: " << gcd << endl;
   cout << "Coefficients (x, y): " << x << ", " << y << endl;
```

```
    int modInverse = multiplicativeInverse(a, b);
    cout << (modInverse != -1 ?
        "Multiplicative inverse of " + to_string(a) + " mod " + to_string(b) + " is: " +
to_string(modInverse)
        : "Multiplicative inverse doesn't exist.") << endl;

    return 0;
}
```

3.RAILFENCE CIPHER:

```
#include <iostream>
#include <string>
using namespace std;

// Encrypt function
string encryptMsg(const string &msg, int key) {
    string encrypted = "";
    for (int i = 0; i < key; ++i) {
        for (int j = i, step1 = (key - i - 1) * 2, step2 = i * 2, toggle = 1; j < msg.length(); toggle = !toggle)
            encrypted += msg[j], j += (step1 && step2) ? (toggle ? step1 : step2) : step1 + step2;
    }
    return encrypted;
}

// Decrypt function
string decryptMsg(const string &encrypted, int key) {
    string decrypted(encrypted.length(), '\0');
    for (int i = 0, idx = 0; i < key; ++i) {
        for (int j = i, step1 = (key - i - 1) * 2, step2 = i * 2, toggle = 1; j < encrypted.length(); toggle =
!toggle)
            decrypted[j] = encrypted[idx++], j += (step1 && step2) ? (toggle ? step1 : step2) : step1 +
step2;
```

```
    }
    return decrypted;
}

int main() {
    string msg, enMsg, decMsg;
    int key;

    cout << "Enter message to be encrypted: ";
    getline(cin, msg);
    cout << "Enter the key: ";
    cin >> key;

    enMsg = encryptMsg(msg, key);
    cout << "\nEncrypted Message: " << enMsg << endl;

    cin.ignore(); // Clear buffer
    cout << "\nEnter message to be decrypted: ";
    getline(cin, enMsg);
    cout << "Enter the key: ";
    cin >> key;

    decMsg = decryptMsg(enMsg, key);
    cout << "\nDecrypted Message: " << decMsg << endl;

    return 0;
}
```

4.RSA:

```
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <cstring>
using namespace std;

long gcd(long a, long b) { return b ? gcd(b, a % b) : a; }
bool isPrime(long num) { for (long i = 2; i <= sqrt(num); i++) if (num % i == 0) return false; return num > 1; }
long modExp(long base, long exp, long mod) {
    long res = 1;
    while (exp) {
        if (exp % 2) res = (res * base) % mod;
        base = (base * base) % mod;
        exp /= 2;
    }
    return res;
}

int main() {
    long p, q, n, phi, e, d;
    char text[50];
    long cipher[50];

    cout << "Enter text: ";
    cin.getline(text, sizeof(text));
    int len = strlen(text);

    do { cout << "Enter prime p: "; cin >> p; } while (!isPrime(p));
    do { cout << "Enter prime q: "; cin >> q; } while (!isPrime(q));
```

```
    n = p * q, phi = (p - 1) * (q - 1);
    do { e = rand() % phi; } while (gcd(phi, e) != 1);
    do { d = rand() % phi; } while ((d * e) % phi != 1);

    cout << "Public key: (" << n << ", " << e << ")\nPrivate key: (" << n << ", " << d << ")\n";

    for (int i = 0; i < len; i++) cipher[i] = modExp(text[i], e, n);
    cout << "Encrypted: "; for (int i = 0; i < len; i++) cout << cipher[i] << " "; cout << endl;

    for (int i = 0; i < len; i++) text[i] = modExp(cipher[i], d, n);
    cout << "Decrypted: " << text << endl;

    return 0;
}
```

5.DEFFIE HELLMAN:

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
    int p, g, a, b;
    cout << "Enter prime number (p): ";
    cin >> p;
    cout << "Enter primitive root (g): ";
    cin >> g;
    cout << "Enter private key for Alice (a): ";
    cin >> a;
    cout << "Enter private key for Bob (b): ";
    cin >> b;
    int A = (int)pow(g, a) % p; // Public key of Alice
    int B = (int)pow(g, b) % p; // Public key of Bob
```

```
    int Ka = (int)pow(B, a) % p; // Shared key (Alice's computation)
    int Kb = (int)pow(A, b) % p; // Shared key (Bob's computation)
    cout << "Shared Key: " << Ka << endl;
    return 0;
}
```